Inter-Domain Management between CORBA and SNMP :

WEB-based Management - CORBA/SNMP Gateway Approach

Subrata Mazumdar

Network and Service Management Research Department

Bell Laboratories,

101 Crawfords Corner Road, Holmdel, NJ-07733

e-mail: mazum@research.bell-labs.com

ABSTRACT:

Common Object Request Broker Architecture (CORBA) is defined by the Object Management Group (OMG) to provide a common architectural framework for object-oriented applications. With CORBA, users gain access to information transparently, without them having to know what software or hardware platform it resides on or where it is located on an enterprises' network. CORBA is also is being adopted for web browser using IIOP and also mapping its Interface Definition Language (IDL) to Java language.

In this paper, we compare two (WBEM and JMAPI) main proposed WEB-based management approaches and show how CORBA/SNMP gateway approach complements the JMAPI approach. In this paper, we also describe the design and implementation of gateway between management application in CORBA domain and agent in SNMP domain. The main function of the gateway is to dynamically convert the method invocations on object references in CORBA domain to SNMP messages for MIB entries at remote agents. We also present an overview of SNMPv2 MIB specification to CORBA-IDL translation algorithm which is currently being standardized by X/Open-Network Management Forum (NMF) sponsored Joint Inter-domain Management (XoJIDM) task force. We also show how to map a variable name in SNMP domain to corresponding object references and attribute name in CORBA domain and vice-versa.

1. Introduction

Common Object Request Broker Architecture (CORBA) [4] is defined by the Object Management Group (OMG) to provide a common architectural framework for object-oriented applications. With CORBA, users gain access to information transparently, without them having to know what software or hardware platform it resides on or where it is located on an enterprises' network. OMG has also defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction among Object Request Brokers (ORB) from different vendors. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system

aspects that are not part of an object's interface. The ORB also provides inter-operability between applications on different machines in heterogeneous distributed environments. In addition, Common Object Services Specification (COSS) [5] defined by OMG provides operating system independent functionalities for life-cycle, naming, event distribution, persistence, etc.

The main benefits of the ORB-based object interaction are: static and dynamic method invocations; multiple high-level language (Java/C/C++/Smalltalk) binding; local/remote transparency - ORB provides location, network protocol and Operating System transparencies.

Our goal is to use CORBA and its service specifications as base technology for implementing management applications and agents and also provide inter-operability with existing SNMP [1] based management applications and agents through dynamic bridging. The main benefits of implementing SNMP MIB and management applications using CORBA are:

- portability of application across multiple network management platforms;

- create a set of reusable class library of SNMP-based managed resources;
  - take advantage of already standardized MIBs defined for network elements;

- reduce the knowledge and skill needed to implement SNMP agents/applications;

The main requirements of implementing SNMP agent and management applications in CORBA domain are translation of SNMP MIB specification to CORBA IDL and a gateway between SNMP domain with CORBA domain and. In this paper, we are going to describe our design and implementation of a gateway between management application in CORBA domain and agents in SNMP domain. The main function of the gateway is to translate the method invocation on an object references of MIB entries to SNMP messages to remote SNMP agent.

In Section 2. we will describe proposed standards for WEB-based management and describe how the CORBA/SNMP gateway approach plays role in development of management applications. In Section 3., we describe inter-operability scenario between SNMP and CORBA domain using our gateway approach. In Section 4., we give an overview of SNMPv2 MIB specification to CORBA-IDL translation algorithm that is currently being specified by XoJIDM task force. In Section 5., we show how to map variable name in SNMP domain to object name and attribute name in CORBA domain. In Section 6., we describe our implementation of a gateway prototype between manager in CORBA domain and agent in SNMP domain.

2. WEB based Management

In this section we compare the approaches proposed by Web based enterprise management (WBEM)[10] group and Java management API (JMAPI) [2]. We also describe how the CORBA/ SNMP gateway approach benefits the JMAPI approach.

2.1 Analysis of WBEM and JMAPI approaches

WEB based enterprise management (WBEM) proposed by proponents of WINTEL platform (Microsoft, Intel, Compaq, BMC, etc.) is proposal for a new object model, called HyperMedia Management Schema (HMMS), a new management protocol, called HyperMedia

Management Protocol (HMMP), and finally OLE/COM based HyperMedia Object Manager (HMOM). HMOM also going to acts as a gateway between existing SNMP agents and DMTF's DMI enabled work stations, hardware and software components.

WBEM approach is based on a set of new proposals - a new object model, a new management protocol on top of HTTP and OLE/DCOM as network management platform. WBEM looks like full of promise because none of its component exist today. There is also no mapping of SNMP MIB to IDL/ODL of DCOM yet. The HMTP of WBEM will also depend on HTTP server and thus we will see more traffic through HTTP server. This approach is in contrast with integration of web browser with WEB-NFS, CORBA/IIOP or OLE/DCOM approaches such that load on HTTP server is reduced.

Java management API (JMAPI) proposed by Javasoft and supported by its partners (Cisco, Bay Networks, 3COM etc.). JMAPI provides two new approaches, direct download of Java code and Java Remote Method Invocation (RMI) based managed object framework, and Java interfaces for SNMP protocol stack. Direct download of Java code is for future Java-OS based appliances.

JMAPI would requires us to use either java-based management application or SNMP-based application. There are some problems with these approach - JMAPI does not say how to map SNMP MIB to Java. An API to SNMP protocol stack is not consistent with the class-based and typed based programming approach of Java. SNMP is a protocol and does not conform to distributed type-safe programming and Java class libraries to SNMP protocol will not solve that problem.

Managed object framework of JAMPI requires one to implement 4-5 classes where as the same result can be obtained by mapping MIB->IDL->Java (SUN's JOE framework) and implementing only one class as server side object. RMI is only applicable if the server side objects are java based object. Managed object framework and RMI usage also do not specify how they can use existing MIB.

Both WBEM and JMAPI is a ground up effort to establishes their sponsors' technology as standard technology for network management. In both cases support for SNMP is after thought. Since SNMP is not going to be replaced overnight by either of these two protocols, both approach need to specify exactly how they are going to support SNMP MIB in their object models.What these two approaches should do is start from SNMP MIB and show how their approach maps MIB to their object model and how dynamic protocol translation is going to work.

2.2 Benefits of CORBA/SNMP Gateway approach

The CORBA/SNMP gateway approach is based on the specification for inter-operable management between CORBA, OSI and SNMP domains. The specification is defined by XoJIDM group sponsored by X/Open and Network Management Forum and this effort is four year old effort. The XoJIDM specification based CORBA/SNMP gateway approach complements the JMAPI approach. The main reason for this that XoJIDM specification maps SNMP MIB specification to OMG-IDL and one can use mapped IDL to generate Java classes that can be used with JMAPI classes. Although one can use CORBA<->OLD/DCOM bridge to cross the domains and the bridging solutions is good at infrastructure level, it does not provide a clear idea how an

3

SNMP MIB maps at programming level API on OLE/DCOM domain.

The main benefit of CORBA/SNMP Gateway approach is that applications developers do not have to know SNMP to manage remote devices. This SNMP->IDL->Java mapping will complement the JMAPI by providing device specific Java classes that can be used to interact with managed devices at remote agent. Java classes obtained using MIB->IDL->Java translation would define the stubs needed for remote access. In addition all the tools to develop applications using MIB->IDL->Java translation is freely available and part of open standard.

There are other advantages of CORBA/SNMP approach. The information at MIB of remote agent are part of enterprise wide directory service and SNMP MIB would appear as yet another information source. Although the access to SNMP MIBs are dependent on the location agent, CORBA/SNMP gateway would provides location transparencies through CORBA ORB. facilities. The object references to the entries of MIB table can be easily passed around in manager-to-manager communication. The gateway itself can be deployed as component of Internet/Intranet Web-server family and the CORBA/SNMP gateway can be configured as trusted SNMP manager that interacts with agent and security feature of Java/CORBA can be used to control access.

Based on the promise of Java based NET-PC and JavaOS based appliances, MIB->IDL->Java mapping can be used to implement MIB for embedded system. Finally, the CORBA/SNMP gateway can also be easily extended to support DMTF's DMI through remote MI interface and through the MIF->MIB-IDL mapping.

3. Inter-operability between CORBA and SNMP Domains

Figure 1. describes the interaction between CORBA based Manager and existing SNMP agent using CORBA->SNMP gateway. The CORBA->SNMP gateway converts the IDL interface based get/set operation using object references to SNMP Request messages and converts the SNMP result messages as return value of the operation. How the operations are mapped to SNMP message PDUs are implementation dependent. The CORBA->SNMP gateway also monitors the SNMP Trap/Notification port, and converts the SNMP notification messages to OMG Notification service based events. The CORBA-based manager subscribes for notification with the event service.
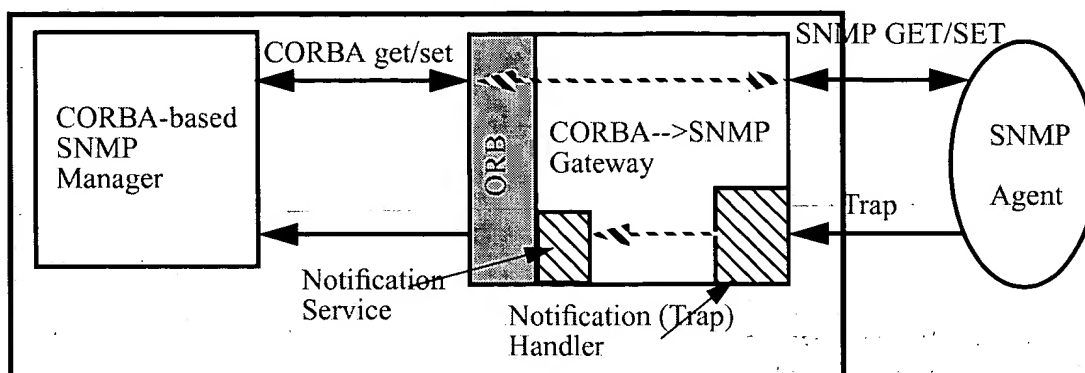

Figure 1. The interaction between Gateway and CORBA-based SNMP Manager.

Figure 2. describes the interaction between existing SNMP manager and CORBA based SNMP MIB implementation using SNMP->CORBA gateway. This approach follows the protocol independent network management architecture presented in [6].The gateway maps SNMP messages to CORBA-IDL operations. When the CORBA based SNMP agent receives an SNMP PDU from SNMP protocol stack, it will convert the message into IDL type and then interact with the CORBA based object server. Since an SNMP message PDU can specify more than one variable that may span multiple table and rows, each SNMP PDU may potentially generate more than one CORBA get/set operation. The gateway also maps the notifications(events) generated by objects in CORBA-based agent to SNMP notifications.
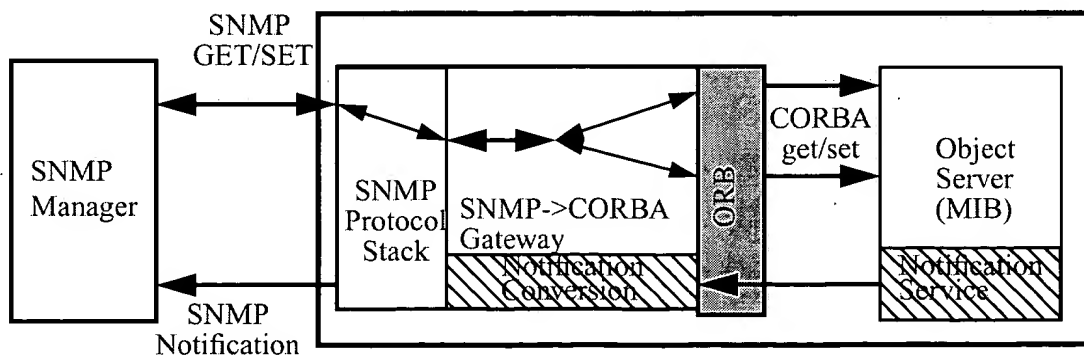

Figure 2. The components of CORBA based SNMP Agent.

The main tasks of the SNMP/CORBA gateway are as follows: map SNMP variable name to combination of object reference and attribute name and vice-versa; map SNMP notification messages to COSS event services based events.

4. SNMPv2 Specification to CORBA-IDL Translation

In this section, we describe only the mapping of groups and tables in SNMP MIB specification to CORBA IDL interfaces[11], [7]. This mapping is currently being standardized by the X/Open and NM Forum sponsored Joint Inter-Domain Management (XoJIDM) task force. See [11] for detailed description of mapping of mapping of ASN.1 types to IDL types and SMI macros to IDL interface.

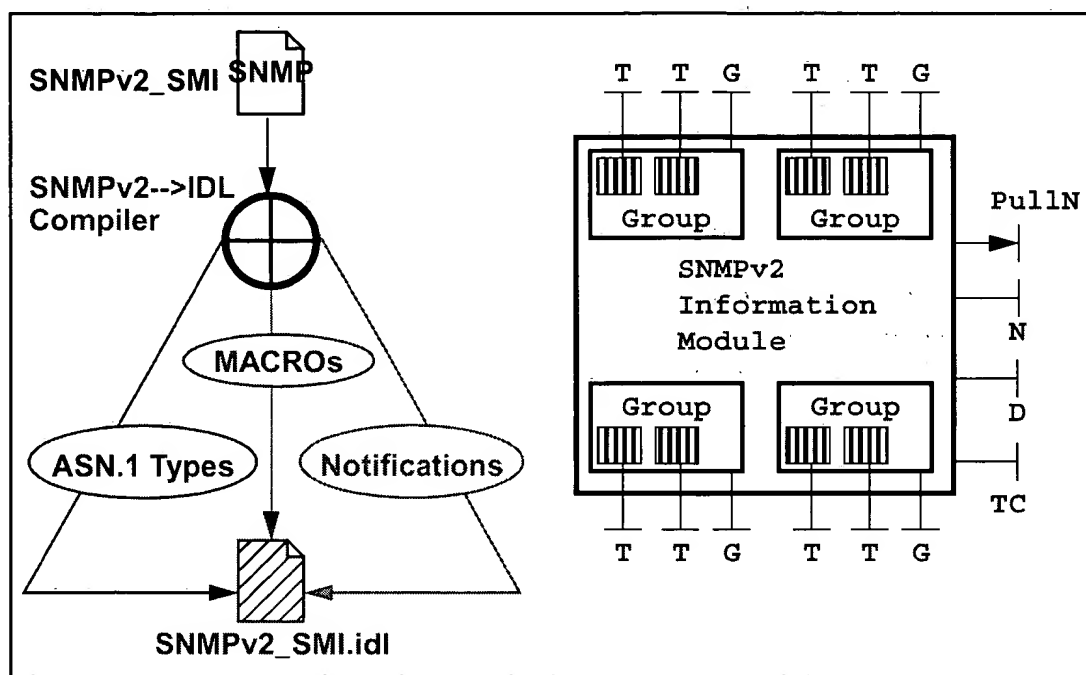The outline of translation of SNMP information module to CORBA-IDL is as follows (as shown in Figure 3.) :

Figure 3. The mapping of SNMP Information Module to IDL.

1.   map an SNMP Information module (SMI document) into an IDL module;

 • all interfaces, types and constant generated from a SNMP information module will be within the scope of the corresponding IDL module

 • declare the imported types in the information module as typedef of imported IDL type

 • declare two IDL interfaces, called SnmpNotifications and PullSnmpNotifications (N & PullN in Figure 3.), if there is atleast one NOTIFICATION-TYPE macro in the SNMP Information module.

6

`SnmpNotifications` interface will be used for typed-push event communication and `PullSnmpNotifications` interface will be used for typed-pull event communication.

- declare a pusedo IDL interface, called `DefaultValues` (D in Figure 3.), if there is atleast one OBJECT-TYPE macro with DEF-VAL clause; Pseudo IDL interafces generate library code similar to specification of CORBA::TypeCode interafce.

- declare a pusedo IDL interface, called `TextualConventions` (TC in Figure 3.), if there is atleast one TEXTUAL-CONVENTION macro with DISPLAY-HINT clause;

2. map each of the ASN.1 type into IDL type using the translation scheme defined in [11]. A complex ASN.1 data type (used to describe PDUs for SNMP) may generate more than one IDL data type;

3. map the value-specification of SYNTAX clause of TEXTUAL-CONVENTION macro, such as DisplayString, into an IDL type.

- if DISPLAY-HINT cluase is present then, define to two operations within the scope of `TextualConventions` interface for converting the typed value to string and string to typed value.

4. map the value of the invocation of the MODULE-IDENTITY macros into a constant IDL literal of type ASN1_ObjectIdentifier;

5. map the value of the invocation of the OBJECT-IDENTITY macros into a constant IDL literal of type ASN1_ObjectIdentifier;

6. each group is mapped as follows: one IDL interface is generated for the group (G in Figure 3.)and one IDL interface (T in Figure 3.) is generated for entries of the each of the table in the group;

- non-tabular variables of a group are mapped as IDL attribute within the scope of the IDL interface for the group; column variables of the tables are mapped as attributes within the scope of the IDL interface for the entries of the table;

  - use the descriptor of the OBJECT-TYPE macro of the corresponding variable as the identifier of attribute;

  - acquire the IDL type of the attribute from the SYNTAX clause of the OBJECT-TYPE macro of the corresponding variable;

  - acquire the mode of the attribute from the MAX-ACCESS clause of the OBJECT-TYPE macro of the corresponding variable;

- if DEFVAL cluase is present then, define an operations that returns the default value in typed form and the operations is defined within the scope of `DefaultValues` interface.

7. map the value of the invocation of the NOTIFICATION-TYPE macros into a constant IDL literal of type ASN1_ObjectIdentifier;

- map the value-specification of OBJECTS clause to an IDL struct which has one

item for each variable in the OBJECT clause; the type of the items of the struct are defined as name-index-value triplet and the type of the value is derived from the variable being mapped.

- define an operation within the scope of SnmpNotifications interface of this module for typed-push event communication; The "in" parameter of the operation are: source SNMP party OID, context OID, time-stamp of the event, and the IDL struct defined for the value-specification of the OBJECTS clause. Use the descriptor of the macro as the identifier for the operation.

- define two operations (try_<op> and pull_<op>) within the scope of PullSnmpNotifications interface of this module, where <op> is the identifier of the corresponding operation in SnmpNotifications interface; The parameters of the operations are same as push-event model but as "out" parameter;

8. assign OID of macros as RepositoryId (using #pragma ID declaration) to all IDL identifiers that corresponds to macro descriptor;

9. ignore the macros related module compliance rules.

Table 1. describes the mapping of the Table Entry, *evalEntry*, to its corresponding IDL interface, evalEntry. The evalEntry interface is defined as subtype of base interface called SNMPMgmt::SmiEntry. The descriptors in the ASN.1 type EvalEntry are used to identify the IDL attributes for the evalEntry interface. A detailed description of MIB to IDL mapping and examples can be found in [11].

Notice that we have used #pragma ID declaration to assign the unique OID of an SNMP variable as the RepositoryId in the Interface Repository. The Interface Repository is a component of ORB and it provides for the storage, distribution, and management of a collection of related objects' interface definitions. Table 2. describes the how to derive the scoped names in IDL from the OID of a variable. Given an OID of a variable, we can easily derive the OID of the group or the table entry of the variable by dropping the last number of the OID. Then we can use the OID as RepositoryId of the group or table entry to get the scoped name of the corresponding interface from the Interface Repository.

8

SNMP OBJECT- TYPE MACRO	FIZ-MIB DEFINITIONS ::= BEGIN
	
	...
	evalEntry OBJECT-TYPE
	    SYNTAX EvalEntry
	    MAX-ACCESS not-accessible
	    STATUS current
	    DESCRIPTION "An entry The (conceptual row) in the evaluation table."
	    INDEX { evalIndex }
	    ::= {evalTable 1 } -- OID of evalTable is assumed to be 1.3.6.1.3.555.2.2
	EvalEntry ::= SEQUENCE {
	    evalIndex Integer32, evalString DisplayString,
	    evalValue Integer32, evalStatus RowStatus
	}
	END

IDL Interface	```
	module FIZ_MIB {
	.....
	interface evalEntry : SNMPMgmt::SmiEntry {
	    #pragma ID evalEntry "1.3.6.1.3.555.2.2.1"
	    /* INDEX : { evalIndex } */
	    readonly attribute Integer32Type evalIndex;
	    #pragma ID evalIndex "1.3.6.1.3.555.2.2.1.1"
	
	    readonly attribute DisplayStringType evalString;
	    #pragma ID evalString "1.3.6.1.3.555.2.2.1.2"
	
	    readonly attribute Integer32Type evalValue;
	    #pragma ID evalValue "1.3.6.1.3.555.2.2.1.3"
	
	    readonly attribute RowStatusType evalStatus;
	    #pragma ID evalStatus "1.3.6.1.3.555.2.2.1.4"
	};
	}; // End of FIZ_MIB module
	``

Table 1. Mapping of a Table Entry, evalEntry, in SNMP to IDL interface.

Document Scoped Macro Label	Variable OID in SNMP	Scoped Name of Interface	Scoped Name of Attribute
FIZ-MIB.evalIndex	1.3.6.1.3.555.2.2.1.1	::FIZ_MIB::evalEntry	::FIZ_MIB::evalEntry ::evalIndex
FIZ-MIB.evalStatus	1.3.6.1.3.555.2.2.1.4	::FIZ_MIB::evalEntry	::FIZ_MIB::evalEntry ::evalStatus

Table 2. Mapping of OID of a variable to the scoped names of the interface and the attribute.

9

5. Naming and Accessing of Objects that Supports MIB-based IDL interface

In this section we will describe how to name object instances of SNMP MIB implementation in CORBA domain and access them based on their corresponding SNMP name. In order to map SNMP names of variable instances to attributes of objects of MIB implementation, we need to define a name tree based on CORBA name service specification [5]. Figure 4. illustrates a possible implementation of the name tree (see [6] detailed description on the configuration of this tree). The basic hierarchy of the name tree is as follows: SNMP-Root, host, fully scoped name of the IDL interfaces for Group/Table/Table Entry generated from SNMP MIB, and finally the instances of objects of corresponding IDL interfaces. In the following subsections, we will give an overview of CORBA naming service specification and then describe how to build a name tree using CORBA naming service for SNMP MIB and finally show how to map SNMP name to names in CORBA domain and then retrieve value of a variable using SNMP name.
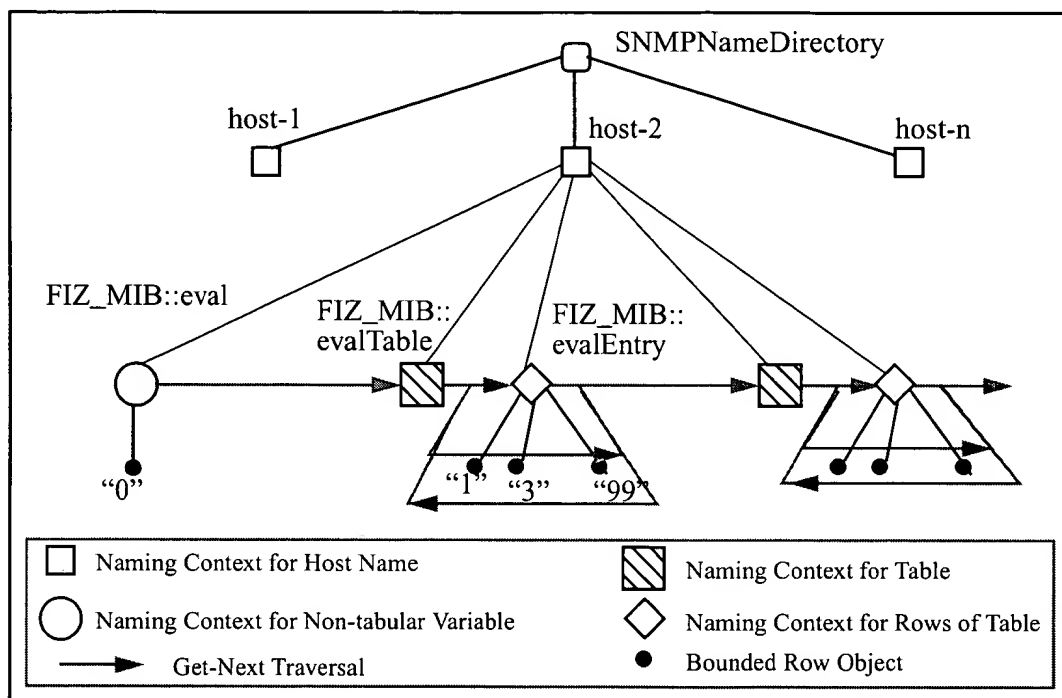

Figure 4. The NamingContext Hierarchy for SNMP MIB at a Specific Agent.

5.1 Overview of CORBA Naming Service Specification

CORBA Naming service specification [5] defines how to associate a name to an object (called name binding) and it closely resembles the naming of files and directories in UNIX file system. A name binding is always defined relative to a "naming context." A naming context is an object that contains a set of name bindings in which each name is unique. Naming context can also be bound to a name. Different name can be bound to an object in the same or different contexts at the same time. To resolve a name is to determine object associated with the name in a given context. A name is an ordered sequence of name components. A name component consists of two el-

10

ements: id and kind, as shown in Table 3. A name with multiple components is called compound name. Table 3. describes data types and interfaces needed to support CORBA naming service. NamingContext interface supports the operations needed to create a new naming context, bind a name to an object, unbind a name, resolve a name to an object and listing of all the names bound to the naming context.

```
Module CosNaming {
   typedef string Istring;
   struct NameComponent { Istring id; Istring kind; };
   typedef sequence<NameComponent> Name;
   .......
   interface NamingContext { ...... };
};
```

Table 3. Data types and interfaces of CORBA Naming Service Specification.

In the following discussion we will only use the id attribute and ignore the kind attribute (by way of initializing it with zero-length string)

5.2 Building of Name Tree of SNMP MIB using CORBA Naming Service

Each of the SNMP host (or entity that represent an agent) will be represented by a naming-context (within the scope of SNMP root, called SNMPNameDirectory). The naming-context for host will contain all the objects of MIB implementation at the host. The id of the name-component of the naming context will initialized by the host name.

Within the scope of each host, there will be a naming-context for each one of the group, table, and table entry which represent all the resources of the SNMP MIB implementation at the host. The naming context for interface type are named using the fully scoped name of the corresponding IDL interface.

The objects (instance of objects that supports IDL interfaces obtained by mapping SNMP MIB) of the MIB implementation are bound within the context of the corresponding interface type. The names of the bounded object are the instance information needed to identify all the variables of row of conceptual table. For non-tabular variables of group, the name is always zero("0") and for rows of tabular variable, the instance information depends on the INDEX clause of the corresponding table. The bounded objects are the leaf-object of the SNMP name tree. In order to support get-next traversal, the naming-contexts for the interface types need to be extended in order to retrieve of the objects in the SNMP lexicography order. The arrows in Figure 4. represents the name-tree traversal for get-next operation.

5.3 Accessing of Values of Variables Using Objects bound to SNMP Name Tree

Table 4. describes how to map the SNMP name of an instance of non-tabular variable of a group to CORBA naming service based names of a MIB object and the corresponding attribute. Given a non-tabular name in OID form, first we have to separate the instance information from variable OID, as shown in second line. Next, we derive the OID of the group by dropping the right-most identifier for the variable (as shown in third line). The group and variable OIDs are

converted to corresponding IDL scoped names (as described in Table 2.). Next we use the group name and instance information to derive a compound name of the group object within the scope of naming context for the host.

```
evalSlot.0 ( => 1.3.6.1.3.555.2.1.0)
   => 1.3.6.1.3.555.2.1,   0
   => 1.2.3.4.5.2,     0,    1.3.6.1.3.555.2.1
   => FIZ_MIB::eval,   0,     FIZ_MIB::eval::evalSlot
   => <FIZ_MIB::eval,  0> ,    FIZ_MIB::eval::evalSlot
```

Table 4. Mapping of Non-tabular variable name to CORBA name and IDL attribute.

In Table 4., we show how to access an instance of evalSlot.0 (which is actually represented by 1.3.6.1.3.555.2.1.0) to the IDL scoped name of the corresponding group interface (FIZ_MIB::eval) and attribute (FIZ_MIB::eval::evalSlot) of the interface. The compound name of the object (within the context of the host) that supports eval is derived by mapping the fully scoped name of the interface and the instance information ("0") to id part of name-components of the compound name. Given the compound name, we can get the reference to the object that represents the eval group by using resolve() operation on the host naming context. Then we can use the resolved object reference to retrieve the value of the variable by invoking get operation associated with the attribute, FIZ_MIB::eval::evalSlot.

```
evalStatus.2 ( => 1.3.6.1.3.555.2.2.1.4.2)
   => 1.3.6.1.3.555.2.2.1.4, 2
   => 1.3.6.1.3.555.2.2.1,   2,         1.3.6.1.3.555.2.2.1.4
   => FIZ_MIB::evalEntry, 2, FIZ_MIB::evalEntry::evalStatus
   => <FIZ_MIB::evalEntry, 2>, FIZ_MIB::evalEntry::evalStatus
```

Table 5. Mapping of Tabular variable name to CORBA name and IDL attribute.

Table 5. describes how to map the SNMP name of an instance of tabular variable of a conceptual table to naming service based names of a MIB object and the corresponding attribute of the interface of the object. In Table 5., we first derive the OID of the variable (1.3.6.1.3.555.2.2.1.4) and its instance information (2) from the variable name (1.3.6.1.3.555.2.2.1.4.2). Then, we use the OID of the variable to get (by dropping the last number of the OID) the OID (1.3.6.1.3.555.2.2.1) of the corresponding Table Entry. Then we map the OIDs of table entry and the variable to corresponding IDL Scoped name of the interface (FIZ_MIB::evalEntry) and attribute (FIZ_MIB::evalEntry:: evalStatus), respectively. Then we use IDL scoped name of the IDL interface for the table entry and the instance information to derive the compound name (<FIZ_MIB::evalEntry, 2>), within the scope of the host context, of the object representing the row in the table. Finally, we can use the object reference, obtained by resolving the compound name with respect to naming context for the host, to retrieve the value of the attribute, FIZ_MIB::eval Entry::evalStatus, by invoking the get operation associated with the attribute.

TABLE 6. describes how to resolve the name of object (as described above) to the corresponding object-reference. First, we need to find the reference to the root for SNMP name tree. Then we find the reference to the naming context for the host. Then we form a compound name (of type Naming::Name) and its components are (in order): interface type name, and the

index information of the variable. Then, we invoke the resolve() operation using naming context of host using the compound-name. If there is an object bound with that name, then an object reference of the MIB entry will be returned. Although we have shown typed operation to get the value of the variable, in reality we have to use Dynamic Invocation Interface (DII) of ORB to invoke the get operation associated with attribute for the variable. Note that the above scheme is also consistent with the way one would resolve the name of object of any other IDL interface in CORBA-only environment. Once an object-reference is obtained, an application programmer does not have to worry about index information.

```
CosNaming::NamingContextRef snmp_name_directory;
CosNaming::NamingContextRef host_naming_context;
CosNaming::Name cos_name;
CosNaming::NameComponenet entry_cos_name_comp[3];
CORBA::ObjectRef obj;
    snmp_name_directory = XXX_get_SNMP_name_directory();
    // get the naming-context using the host
    entry_cos_name_comp[0].id("host-2");
    entry_cos_name_comp[0].kind("");
    cos_name(1, 1, entry_cos_name_comp);
    obj = snmp_name_directory->resolve(cos_name);
    host_naming_context = CosNaming::NamingContext::_narrow();


FIZ_MIB::evalRef eval_ref;
ASN1_Integer eval_slot;
    // get the object reference to eval group using interface type and index
    entry_cos_name_comp[0].id("FIZ_MIB::eval");
    entry_cos_name_comp[0].kind("");
    entry_cos_name_comp[1].id("0");
    entry_cos_name_comp[1].kind("");
    cos_name(2, 2, entry_cos_name_comp);
    obj = host_naming_context->resolve(cos_name);
    // get the value of the attribute
    eval_ref = FIZ_MIB::eval::_narrow(obj);
    eval_slot = eval_ref->evalSlot();


FIZ_MIB::evalEntryRef eval_entry_ref;
ASN1_Integer eval_value;
    // get the object reference of evalEntry of row index 2 using the interface
    // type and index-information
    entry_cos_name_comp[0].id("FIZ_MIB::evalEntry");
    entry_cos_name_comp[0].kind("");
    entry_cos_name_comp[1].id("2");
    entry_cos_name_comp[1].kind("");
    cos_name(2, 2, entry_cos_name_comp);
    obj = host_naming_context->resolve(cos_name);
    // get the value of the attribute
    eval_entry_ref = FIZ_MIB::evalEntry::_narrow(obj);
    eval_value = eval_entry_ref->evalValue();
```

TABLE 6.   **Accessing the value of an attribute of an object using SNMP name.**

13

6. Prototype Implementation of CORBA Manager to SNMP Agent Gateway

Figure 5. describes the components of our implementation of a gateway between management application in CORBA domain and agent in SNMP domain. The prototype is built using SUN's NEO 1.0 [9] on Solaris 2.5 running on SPARCstation. The gateway prototype consists of following components: NEO- DSI, SNMP-DIR, SNMP API library from CMU. We use the NEO's implementation of Dynamic Skeleton Interface (DSI) [4] in order to intercept all operations invoked on the object references associated with the entries in the MIB at remote SNMP agent. DSI is way to deliver request from an ORB to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The DSI allows an ORB to invoke same upcall routine, called Dynamic Implementation Routine(DIR) for all request on particular object. Our implementation of the SNMP-DIR provides a generic implementation of all IDL interface generated from SNMP MIB. The SNMP-DIR dynamically converts an operation invocation on the MIB objects in the CORBA domain into SNMP messages using CMU's SNMP API library. Any changes in the SNMP MIB would only require to re-compile the SNMP MIB to IDL. The gateway does not require the implementation of server side object based on the IDL interface of SNMP MIB.
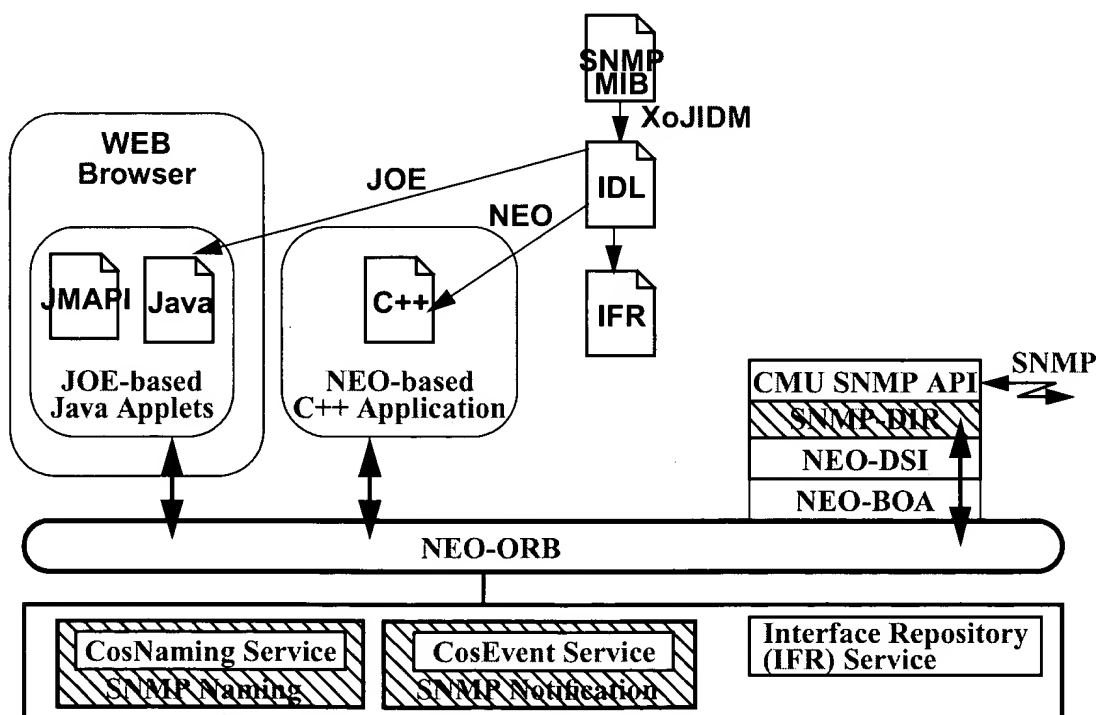

Figure 5. Architecture of prototype implementation of CORBA manager to SNMP agent gateway.

In order to support the MIB name tree described in Figure 4., we have extended the CosNaming service[5] and implemented naming service for SNMP domain. The key feature of this service is that it understands the hierarchy of the name tree shown in Figure 4., dynamically builds, updates and synchronizes the tree with the entries in the MIB of remote SNMP agents. Our implementation of the SNMP naming services extend the list() operation of NamingContext

interface such that it builds the list of entries in table in the lexicography order of SNMP get-next operation. Applications written against SNMP name service do not have to understand SNMP naming, they can be written based on CORBA naming services interface and still access the object references to entries of the MIB at remote agent. We have also extended the CosEvent service to a SNMP notification service in CORBA domain. The SNMP Notification Service provides a built-in SNMP notification/trap handler (as shown in Figure 1.) that converts SNMP notification/trap to un-typed event in CORBA domain.

The client applications can be written Java or C++, like any other application in CORBA domain, invokes get/set methods associated with the attributes of IDL interface for SNMP group/ table. The client applications find object references for entries in the MIB by resolving a compound name consisting of host name, interface type for entry and the row index (as described in Section 5.) in the name tree supported by our SNMP naming service. The client applications only need the IDL definitions generated from the SNMP MIB files. Modification of SNMP MIB specification or additions of new MIB definition would only require the mapping to IDL and then updating the Interface Repository. The SNMP MIB specification is not need both at the client side and at gateway for run-time environment.

6.1 Status of Our Gateway Prototype

We have implemented gateway between management applications in CORBA domain and agent in SNMP domain. At this time (October 1996) , we only support SNMPv1 agent. The gateway is implemented using NEO and CMU's SNMP API. The mapping of CORBA get/set operation invocation of attributes to SNMP GET/SET message primitive is completed. We also dynamically convert SNMP Traps to untyped events in CORBA domain and client applications can subscribe this event. Thus trap handling methods can be written in CORBA.

In addition to C++ based applications, we have also written a set of JMAPI based applets. So far we have written three Java applets using JOE and JMAPI classes: a MIB Browser using BrowserFolder class, an SNMP Table Dump using Table class and a counter value based StripChart using StripChart class.

Finally, our implementation is fully compliant with X/Open-NM Forum sponsored XoJIDM specification for Inter-Domain Management

7. Summary

In this paper we have described how the benefits of CORBA/SNMP gateway for the WEB based management. We have also described how the management application and agent can inter-operate between existing SNMP platform and CORBA based distributed object environment. The key components in this inter-operability is mapping of SNMP MIB specification to CORBA IDL and then dynamically mapping SNMP variable names to IDL interface based object reference. In this paper, we have also described the mapping of SNMP MIB specification to CORBA IDL and the mapping is currently being standardized by the X/Open and NM Forum sponsored Joint Inter-Domain Management (XoJIDM) task force.

We have also described how to map SNMP names to OMG Object Service based names and

15

vice-versa. Our CORBA->SNMP gateway use this name mapping scheme to dynamically map SNMP message to IDL operation and vice-versa. We have also described our implementation of gateway between CORBA manager and SNMP agent. In future, we would like build a gateway between SNMP manager and objects in CORBA domain.

References

[1] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1442, April 1993.

[2] JavaSoft, "Java Management API", Specification 1.0, early access, September, 1996, http://java.sun.com/products/JavaManagement/.

[3] SunSoft, "Client/Server Applications for the Web", September, 1996, http://www.sun.com/solaris/neo/joe/.

[4] Object Management Group, "The Common Object Request Broker: Architecture and Specification", Revision 2.0, July 1995.

[5] Object Management Group, "CORBAservices: Common Object Services Specification", Revised Edition, OMG Document # 95-3-31, March 31, 1995.

[6] Subrata Mazumdar, Stephen Brady, and David Levine, "Design of Protocol Independent Management Agent to Support SNMP and CMIP Queries," Proceedings of The Third ISINM, San Francisco, California, April 1993.

[7] S. Mazumdar, "Translation of SNMPv2 Specification into CORBA-IDL," A report of the Joint X/Open/NM Forum Inter-Domain Management task force, August, 1996.

[8] Subrata Mazumdar, "Translation of SNMP MIB Specification into CORBA-IDL", presented at the January 28-29, 1994 XoJIDM meeting.

[9] SunSoft, "NEO Programming Interfaces References", NEO v1.0, October, 1995.

[10] "Web-Based Enterprise Management Standards Effort", July 17, 1996, http://wbem.freerange.com/.

[11] X/Open, "Inter-Domain Management Specifications: Specification Translation", X/Open Preliminary Specification, Draft, August 9, 1995.